Cuttlefish: A Library For Building Distributed Neural Networks

Abu Kamruzzaman*, Rania Mohammedameen Almajalid†, Yu Hou‡ and Teresa Nicole Brooks§

Computer Science, Pace University

Pleasantville NY

Email: *ak91252p@pace.edu, †ra56319p@pace.edu, ‡yh50276p@pace.edu, §tb93141n@pace.edu

Abstract—This paper will discuss the implementation of Cuttlefish, a library for building configurable, fault tolerant, distributed neural networks. Cuttlefish is built on top of the Distributed TensorFlow library. It uses Docker containers to represent units or neurons in the a neural network. More specifically each container (neuron) will run as a service that can communicate with other neurons in the network. Our neural network will be a n layered network, implemented and configured to run on Amazon Web Service's Elastic Container Service (ECS).

I. INTRODUCTION

Machine learning and more specifically the use of Neural Networks have many applications in both the research and commercial software. Though most machine learning techniques and algorithms employed today were developed over 20 years ago the rise of cheap, powerful processors (GPUs and CPUs) and higher capacity storage has allowed these techniques and algorithms to be used at scale.

In this paper we will talk about the implementation of Cuttlefish, a library for building configurable, fault tolerant, distributed neural networks. We will discuss the technologies used to implement this library which includes the use of clustered Docker containers, where each container represents a neuron in the network, Distributed TensorFlow and Amazon Web Service's Elastic Container Service (ECS).

II. BACKGROUND

In this section we will briefly discuss the technologies used to build Cuttlefish. In later sections we will discuss in more detail how these technologies are used.

1) Docker: Docker is the world's leading software containerization platform [9]. Docker is open source, designed to make it easier to create, deploy, and run distributed applications inside lightweight Linux containers. It provides a way to run applications securely isolated in a container by packaging it and all its dependencies as well as libraries. Docker containers can wrap the software into a self-contained unit with a complete file-system that contains everything needed to run [8]: operating system, code, runtime, system tools, system libraries, and configured system resources, etc. By using Docker containers, we can deploy, and back up a workload regardless of environment quickly and easily more than using virtual machines.

Docker allows to change any application dynamically by adding new capabilities and scaling services which makes containers more portable and flexible to use. Docker also has a mechanism for configuring and spinning up containers that should be clustered together or that are dependencies of one another.

One of the Docker advantages is simplified maintenance which means Docker minimize the efforts and risks which associates with application dependencies. The other advantage is increasing developer productivity because it decreases the time that spent setting up

2) TensorFlow: TensorFlow is essentially a framework for building Deep Learning Neural Networks [7]. It was developed by engineers and researchers working on the Google Brain Team within Google's Machine Intelligence research organization. It is an open source software library for machine learning computation using data flow graphs [7]. Data flow graphs describe mathematical computation with nodes and edges. Each node in the graph represents mathematical operations, while edges represent the relationships between nodes. A computation expressed using TensorFlow can be executed in one or more GPUs or CPUs in a desktop, mobile device, or server with a single API.

The initial open-source release of TensorFlow supports multiple devices (CPUs and GPUs) [7] in a single computer which is called the single-machine or single node implementation, while the distributed version supports multiple devices (CPUs and GPUs) across a cluster of machines.

3) Amazon EC2 Container Service (ECS): Amazon EC2 Container Service (ECS) is high performance container management service which supports Docker containers through Management Console or Command Line Interface. It is highly scalable and allows applications to run easily on a managed cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances. Amazon ECS enables applications to be scaled without any additional complexity from a single container to thousands of containers across hundreds of instances [3]. Amazon ECS allows launch and stop container-based applications with simple API calls.

III. MOTIVATION

The motivation for Cuttlefish was born from the the observation that neural networks at their core are computational graphs and more specifically directed computational graphs,
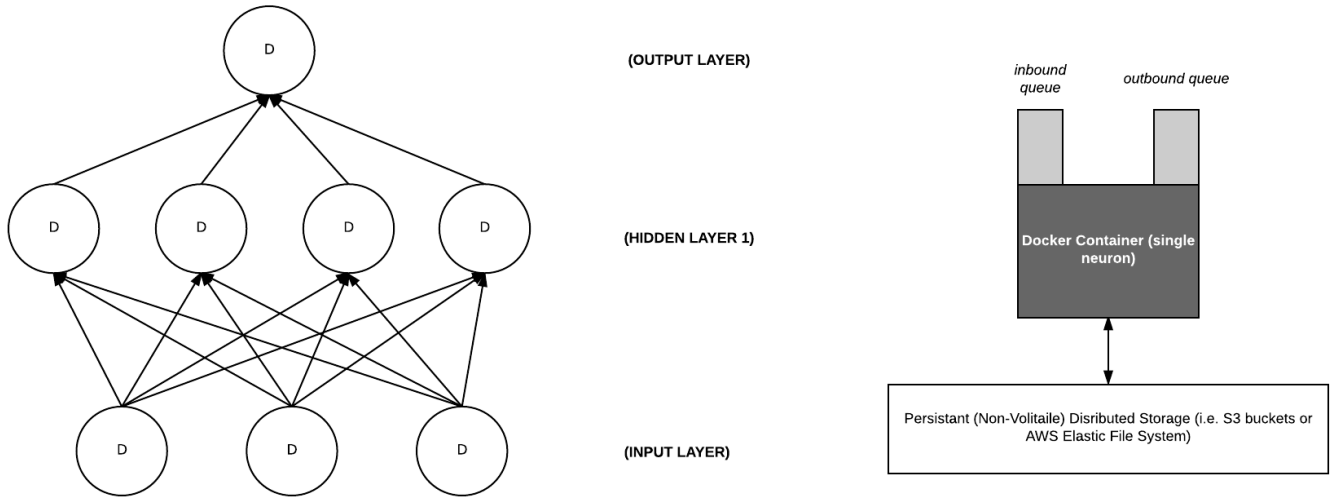
Fig. 1. Diagram example neural network nodes as docker containers. Note, the persistent distributed storage represent non-volatile storage for localized data per docker container. Persistent storage is needed to provide fault tolerance for shared and unshared data.

where each neuron in the graph is a single computation unit. This observation coupled with the understanding that in order to train models and perform inference on new data presented to these models at scale, in a production environment, requires implementing neural networks in a way that enables us to infinitely scale the execution of the these computation graphs in way that allows us to leverage all available system resources.

Tasks for training models and performing inference in neural networks are inherently parallelizable. Hence, it should be possible to implement a library that allows users to create a fault tolerant, distributed neural network, using configurable hyperparameters to drive the dynamic creation of a directed graph of neurons (a neural network). To implement this library we need a way to represent each node in the network physically; as a device or machine where computation execution will take place. We also need a centralized database to store intermediate values for the weight parameters each neuron calculates during model training. Moreover, the neurons in the network would need an efficient mechanism to communicate and pass data from one neuron to another. Lastly, we would need a mechanism to automate configuring and "spinning up" a neural network. Below we briefly discuss the technologies and approaches needed to implement such a library.

1) Physical Representation of Neurons - Docker Containers: The use of Docker to represent a physical neuron was an natural choice because Docker containers are self contained units that enable you to provide everything an application would need to run and nothing more; this includes an operating system, file system (volatile file system), and any needed software, frameworks or tools. Using Docker's Compose tool [10] containers can easily be configured as dependencies of one another, allowing you to create a cluster of containers that can be spun up together and that can communicate with one another.

2) Centralized Storage Of Weighted Model Parameters - Distributed NoSql Database: For our centralized parameter store, because the data is not highly relational and we would need a scalable solution distributed key, value stores such a Amazon Web Services' DynamoDB would be a good fit.

3) Interneuron Communication: For inter neural network communication we would need an Inbound and output queue for each neuron in the network in order to pass data such as computed weights, and training model data from node to node. To implement such communication fault tolerant message broker technologies such as rabbitmq [6] or Apache Kafka [1] are good choices.

4) Automation of "Spinning" Up Elastic Neural Networks: Infrastructure orchestration software such as Kubernetes, Mesos and Amazon Web Service's Elastic Container Service are tools used in both test and production environments to automate orchestrations for creating, configuring and managing docker containers, and hence makes them a natural fit for this task.

IV. OUR APPROACH

** Note, this section should really be called "Our Proposed Approach" as is being written before we have implemented Cuttlefish and before we know more about Distributed TensorFlow's capabilities**

During the initial research phase of this project, we discovered Google's Distributed TensorFlow project in spirit was very similar to the initial idea that motivated our research. Distributed TensorFlow allows you to create a cluster of servers, where each worker process task is associated to a server in the cluster.

Distributed TensorFlow though it is primarily used for modeling machine learning computational models is generic

enough to be considered a general purpose distributed computing library. It offers most of the properties we thought were paramount in implementing a library to enable building a distributed neural networks: fault tolerance, a means of sharing common calculated parameters among devices, cross device communication, as well as distributed execution of a computation graph among nodes in a TensorFlow cluster [4]. Hence, we decided to use Distributed TensorFlow as the base for the Culttlefish library.

Note that Cuttlefish defines a docker container as a single server in a TensorFlow cluster. Hence a Cuttlefish distributed neural network is simply defined as TensorFlow cluster.

1) Computation Graph Distribution: TensorFlow uses a greed heuristic algorithm called the "placement algorithm" [11] to determine how a computation graph will be distributed for execution among all available devices. Default supported device types are CPUs and GPUs, there is also a registration mechanism so that users can implement and register their own device types [11]. TensorFlow provides an interface for users to influence how the computation graph is distributed, by allowing them to give "hints and partial constraints" to the algorithm [11].

Cuttlefish, will use the above described functionality along with the configuration of each docker container's system resources (memory, number of CPUs etc) to attempt to force TensorFlow's placement algorithm to map one node in the computation graph to one Docker container. This is will test our the viability of our motivation of using a single Docker container to represent a single computational unit in a distributed neural network (neuron).

2) Configuring Neural Network (Hyperparameters): With Cuttlefish users define the configuration and shape of their neural network's hyperparameters in a yaml file. By defining these parameters as "code", versioning of these parameters is simple, this also allows a user to automate building these configuration files as a tasks in a larger workflow where hyperparameters are being tested for a given set of training data.

3) Automation & Orchestration: Creating Docker Containers As Per Cuttlefish Configuration File: Cuttlefish's "build" functionality will use the Amazon Web Services' (AWS) contain service's API [2] and it's user defined elastic neural network configuration files to configure and build a distributed neural network using a cluster of docker containers (TensorFlow cluster).

With this approach, Cuttlefish takes the paradigm of infrastructure as code and applies it to the configuring and building of a distributed neural network as a cluster of resources readily available for computation tasks.

Note, though we are using AWS' ECS [3] for orchestration for this proof of concept, tools like Kubernetes and Mesos are better choice as they offer finer grain control over configuring resource allocation per container. Fine grain control of such resources would allow for configuring and tuning system resources per neuron type, thus making the required system resources fit the computation task being performed by a particular neuron. This level of control would be useful when implementing neural networks such as convolutional neural networks, where different types of neurons perform different computational tasks and hence are likely to have different resource needs.

4) Data Set: We are using the MINST data set for training distributed neural networks created by Cuttlefish. "The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 example." [5] We choice the MINST data set because it is well documented, there is extensive benchmark data for it and TensorFlow as the full data set in a preprocessed ready to use format.

V. System & Application Architecture

TBD

VI. Results

TBD

VII. Improvements & Possible Future Work

TBD

VIII. Related Work

TBD

IX. Conclusion

TBD

References

[1] Apache kafka product site.
[2] Aws elastic container serice api documentation.
[3] Aws elastic container service product page.
[4] Distributed tensorflow.
[5] Mnist documentation.
[6] Rabbitmq product site.
[7] Tensorflow homepage.
[8] Top three benefits using docker.
[9] What is docker?
[10] Docker. Docker documentation site.
[11] Paul Barham Eugene Brevdo Zhifeng Chen Craig Citro Greg S. Corrado Andy Davis Jeffrey Dean Matthieu Devin Sanjay Ghemawat Ian Goodfellow Andrew Harp Geoffrey Irving Michael Isard Yangqing Jia Rafal Jozefowicz Lukasz Kaiser Manjunath Kudlur Josh Levenberg Dan Mane Rajat Monga Sherry Moore Derek Murray Chris Olah Mike Schuster Jonathon Shlens Benoit Steiner Ilya Sutskever Kunal Talwar Paul Tucker Vincent Vanhoucke Vijay Vasudevan Fernanda Viegas Oriol Vinyals Pete Warden Martin Wattenberg Martin Wicke Yuan Yu Mart n Abadi, Ashish Agarwal and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016.